Menghitung Total Kemungkinan Pengelompokan pada Tugas Besar 2 Aljabar Linear dan Geometri

Mochammad Fariz Rifqi Rizqulloh - 13523069[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*faris361707@gmail.com*, *13523069@std.stei.itb.ac.id*

***Abstrak*—Permasalahan pengelompokan mahasiswa dalam tugas besar kedua pada mata kuliah Aljabar Linear dan Geometri di Teknik Informatika ITB memerlukan pembentukan kelompok yang memenuhi aturan unik, yaitu tidak memperbolehkan mahasiswa yang pernah satu kelompok di tugas pertama untuk kembali berkelompok di tugas kedua. Aturan ini membuat perhitungan total kemungkinan pengelompokan menjadi kompleks. Makalah ini menjelaskan pendekatan efisien untuk menyelesaikan masalah ini menggunakan representasi graf dan algoritma berbasis rekursi. Implementasi dalam Bahasa C++ disertai dengan analisis kompleksitas waktu menunjukkan bahwa metode ini dapat mengurangi redundansi eksplorasi dibandingkan dengan pendekatan brute-force. Selain itu, makalah ini juga mengkaji hubungan antara model matematis dan pemrograman untuk menyelesaikan permasalahan kombinatorik dalam konteks praktis.**

***Kata Kunci*—Rekursi, Bruteforce, Kombinatorika, Optimasi Pemrograman, Memoisasi.**

I. Pendahuluan

Dalam pengujian kemampuan dan pemahaman mahasiswa-mahasiswanya dalam materi yang telah disampaikan, sebagian besar mata kuliah di program studi Teknik Informatika ITB memberikan tugas besar. Tugas besar yang dimaksud merupakan tugas yang dikerjakan secara berkelompok, dengan jangka waktu cukup panjang. Pada mata kuliah aljabar linier dan geometri, mahasiswa-mahasiswi Teknik Informatika ITB diberikan tugas besar sebanyak 2 kali, Dimana tugas pertama rilis pada tanggal 1 Oktober 2024, sementara tugas besar kedua rilis pada tanggal 21 November 2024.

Dalam menentukan anggota kelompok pada tugas besar pertama, para mahasiswa dibebaskan memilih dengan bebas, baik mahasiswa dari kelas lain, maupun dari kelas yang sama, hanya dibatasi bahwa anggota kelompok haruslah mahasiswa yang mengambil mata kuliah aljabar linear dan geometri, serta jumlah anggota pada tiap kelompok adalah 3 orang.

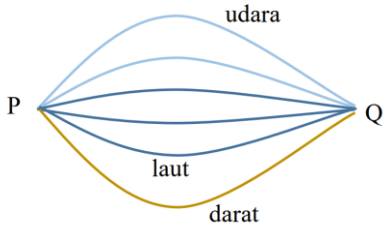
Dalam menentukan anggota kelompok pada tugas besar kedua, diberlakukan system yang sama dengan tugas besar pertama, tetapi dengan 1 syarat tambahan, yaitu tidak boleh ada pasangan mahasiswa yang berada di kelompok yang sama pada tugas besar 1 maupun tugas besar 2. Dengan kata lain, jika A dan B berpasangan pada tugas besar pertama, maka A dan B tidak boleh berpasangan di tugas besar kedua. Adanya syarat tambahan tersebut membuat perhitungan total kemungkinan pengelompokan yang mungkin menjadi jauh lebih kompleks. Salah satu pendekatan naif adalah dengan melakukan *brute-force* semua susunan pengelompokan yang mungkin, yang tentunya cara tersebut sangatlah tidak efisien. Makalah ini akan menelisik cara efisien dalam menyelesaikan permasalahan tersebut sekaligus implementasinya dalam Bahasa Python, serta menelusuri kompleksitas waktu dari program tersebut.

II. Landasan Teori

A. Kombinatorika

Kombinatorika adalah cabang matematika untuk menghitung (counting) jumlah penyusunan objek-objek tanpa harus mengenumerasi semua kemungkinan. [1] Terdapat dua buah prinsip dalam kombinatorika yaitu the *addition principle* (AP) dan the *multiplication principle* (MP). [2] prinsip ini merupakan operasi paling dasar dalam menyelesaikan persoalan-persoalan perhitungan enumerasi.

Kaidah penjumlahan digunakan ketika hendak menghitung jumlah total kemungkinan dari dua atau lebih kejadian atau kondisi yang setiap kondisinya tidak dapat terjadi secara bersamaan. Kaidah penjumlahan secara intuitif mengatakan bahwa, jika ada A buah cara untuk melakukan sesuatu dan B buah cara untuk melakukan hal lain, tetapi tidak dapat melakukan keduanya di saat yang sama, maka ada A + B cara untuk memilih salah satu dari hal tersebut. Contoh aplikasi kaidah penjumlahan adalah dalam persoalan pencarian banyaknya cara seseorang pergi dari kota P ke kota Q jika ada 2 cara untuk melalui jalur udara, 3 cara untuk melalui jalur laut, dan 1 cara untuk melalui jalur darat. Berdasarkan *addition principle*, diperoleh banyaknya cara dari kota P ke kota Q adalah 2+ 3 + 1 = 6 cara



Gambar 2.1. Ilustrasi Persoalan Kaidah Penjumlahan

Adapun kaidah perkalian atau MP menyatakan bahwa jika sebuah kejadian yang dapat dipartisi menjadi k kejadian terurut dengan banyak cara untuk mencapai kejadian ke-i adalah ni untuk setiap $i \in [1, k]$, total banyaknya cara untuk kejadian tersebut dapat terjadi adalah $n_1 \times n_2 \times n_3 \times ... \times n_k$. [1]. Kaidah perkalian digunakan Ketika Digunakan ketika hendak menghitung jumlah total kemungkinan dari dua atau lebih

kejadian atau kondisi yang setiap kondisinya terjadi secara bersamaan. Kaidah perkalian secara intuitif mengatakan bahwa jika ada A buah cara untuk melakukan sesuatu dan B buah cara untuk melakukan hal lain secara bersamaan, maka akan ada A × B cara untuk melakukan hal tersebut. Contoh aplikasinya adalah ketika seseorang ingin pergi dari kota A ke kota D dan harus melewati kota B dan kota C, Dimana terdapat 2 Jalur dari kota A menuju kota B, terdapat 5 jalur dari kota B menuju kota C, dan terdapat 3 jalur dari kota C menuju kota D.

Gambar 2.2. Ilustrasi Persoalan Kaidah Perkalian

Total banyaknya cara untuk pergi dari kota A ke kota D adalah perkalian banyaknya cara dari kota A ke kota B, dari kota B ke kota C, dan dari kota C ke kota D. Sehingga diperoleh total banyaknya cara adalah $2 \times 5 \times 3 = 30$ cara. Prinsip dasar kombinatorial ini akan dijadikan alat untuk memperoleh teknik-teknik kombinatorial lain yang lebih penting seperti permutasi dan kombinasi.

B. Permutasi

Misalkan terdapat n buah objek tidak identik dan r buah kotak dengan $r \leq n$. Banyaknya cara berbeda untuk mengisi setiap kotak sedemikian sehingga setiap kotak tepat memiliki sebuah objek adalah sebagai berikut :

- Banyaknya cara untuk mengisi kotak pertama adalah n.
- Banyaknya cara untuk mengisi kotak kedua adalah $(n - 1)$.
- ...
- Banyaknya cara untuk mengisi kotak ke-r adalah $(n - r - 1)$.

Berdasarkan MP, diperoleh bahwa banyaknya cara mengisi r buah kotak dengan setiap kotak berisi tepat sebuah objek dari n buah objek dan $r \leq n$ adalah $n \times (n - 1) \times ... \times (n - r - 1)$. Bentuk kasus ini disebut sebagai permutasi. Secara umum, permutasi r dari n elemen dinotasikan dengan $P_r^n = \frac{n!}{(n-r)!}$, dimana $x!$ menyatakan factorial dari x, yakni $x! = x \times (x - 1) \times (x - 2) \times ... \times 2 \times 1$. Disamping itu, terdapat konsensus untuk nilai dari $0! = 1$.

C. Kombinasi

Kombinasi adalah bentuk khusus dari permutasi, bedanya dengan permutasi, kombinasi tidak memperhitungkan urutan kemunculan. [9] Misalkan A adalah sebuah himpunan dengan setiap elemennya berbeda satu sama lain. Himpunan-kombinasi dari A didefinisikan sebagai subhimpunan dari A. Lebih khusus, himpunan-kombinasi-r dari A adalah subhimpunan A yang memiliki r elemen. [1]

Kombinasi r dari n adalah banyaknya himpunan-kombinasi-r dari sebuah himpunan dengan n elemen. Untuk mengaitkan kombinasi dengan permutasi, agar memperoleh nilai kombinasi r dari n elemen sesuai definisi, tinjau bahwa banyaknya cara memasukkan n objek ke dalam r kotak adalah P_r^n. Karena urutan objek di setiap kotak dalam definisi kombinasi tidak diperhatikan, diperoleh bahwa kombinasi r dari n elemen adalah P_r^n / P_r^r. Secara matematis,

$$C_r^n = \frac{P_r^n}{P_r^r} = \frac{n!}{(n-r)!\,(r!)}$$

Adapun properti penting dalamkombinasi, yaitu

$$C_r^n = C_{n-r}^n$$

D. Rekursi

Secara simpel, Rekursi adalah sesuatu yang dalam pendefinisiannya terdapat terminologi dirinya sendiri. Algoritma rekursif adalah Algoritma yang terdapat rekursi didalam pendefinisiannya. Algoritma Rekursif didefinisikan oleh dua bagian yaitu:

1. Basis
   Basis terdiri dari nilai fungsi yang terdefenisi secara eksplisit. Sebuah fungsi rekursif akan berhenti memanggil dirinya sendiri di Basis.
2. Rekurens
   Rekurens mendefinisikan fungsi dalam terminologi dirinya sendiri. Berisi kaidah untuk menemukan nilai fungsi pada suatu input dari nilai-nilai lainnya pada input yang lebih kecil.

Salah satu permasalahan yang dapat direpresentasikan dalam algoritma rekursif adalah menghitung nilai faktorial.

$$x! = \begin{cases} 1, & x = 0 \\ n \times (n-1)!, & x > 0 \end{cases}$$

Berdasarkan rumus diatas, diketahui bahwa basis dalam faktorial adalah ketika n = 0, dengan nilai basis 0. Jika kita ingin mencari 5! maka prosesnya akan seperti berikut,

$$5! = 5 \times 4!$$
$$5! = 5 \times 4 \times 3!$$
$$5! = 5 \times 4 \times 3 \times 2!$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1!$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1 \times 0!$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1 \times 1$$
$$5! = 120$$

Fungsi rekursi dapat diimplementasikan ke dalam pemrograman. Contoh implementasi fungsi factorial dengan algoritma rekursif adalah sebagai berikut :

```
function factorial(x : integer) : integer
begin
  if x = 0 then
    factorial := 1
  else
    factorial := x * factorial(x - 1)
end;
```

Gambar 2.3. Pseudocode fungsi factorial dengan rekursi

E. Kompleksitas Waktu Algoritma

Pekerjaan utama di dalam kompleksitas waktu adalah menghitung jumlah tahapan komputasi di dalam algoritma.

Tahapan komputasi diukur, biasanya, berdasarkan banyaknya operasi dasar yang dilakukan dalam algoritma. Ukuran kompleksitas waktu sebuah algoritma ditetapkan berdasarkan besarnya ukuran masukan. Ada tiga macam kompleksitas waktu:

1. $T_{max}(n)$, yaitu kompleksitas waktu untuk kasus terbutuk *(worst case)*, kebutuhan waktu maksimum|

2. $T_{min}(n)$, yaitu kompleksitas waktu untuk kasus terbaik (best case), kebutuhan waktu minimum,

3. $T_{avg}(n)$, yaitu kompleksitas waktu untuk kasus rata-rata (average case), kebutuhan waktu secara rata-rata.

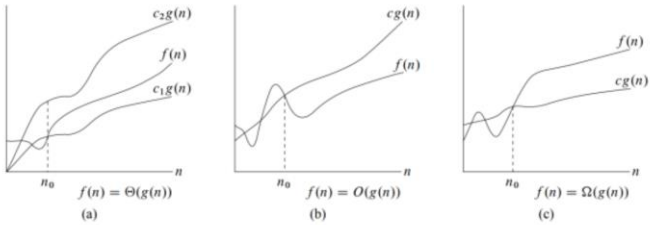
Jenis kompleksitas waktu yang dipertimbangkan dalam makalah ini adalah kompleksitas waktu maksimum.

Representasi kompleksitas waktu biasanya dituliskan dalam bentuk sifat asimtotiknya, terdapat 3 buah fungsi asimtotik, yaitu big-O, big-Omega, dan big-Theta. Masing-masing definisinya adalah sebagai berikut:

1. Big-O, atau $O(f(n))$, adalah fungsi asimtotik atas dari $T(n)$ jika terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \leq Cf(n)\ \forall n \geq n_0$.

2. Big-O, atau $O(f(n))$, adalah fungsi asimtotik atas dari $T(n)$ jika terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \geq Cf(n)\ \forall n \geq n_0$.

3. big-Theta, atau $\Theta(f(n))$ adalah fungsi asimtotik dari $T(n)$ jika $T(n)$ memiliki asimtotik bawah $\Omega(f(n))$ dan asimtotik atas $O(f(n))$.



Gambar 4 (a) Fungsi Asimtotik (b) Fungsi Asimtotik Atas (c) Fungsi Asimtotik Bawah

Pada umumnya, representasi fungsi asimtotik kompleksitas waktu yang digunakan adalah fungsi asimtotik atas atau $O(f(n))$. Properti yang paling umum dari fungsi asimtotik ini adalah sebagai berikut :

1. $O(f(n) + g(n)) = O(max(f(n), g(n))$.

2. $O\big(f(n) \times g(n)\big) = O\big(f(n)\big) \times O\big(g(n)\big)$.

3. $O(c \times f(n)) = O(f(n))$

4. $f(n) = O(f(n))$

5. $O(P(n)) = O(x^n), dengan\ P(n) = a_0x^n + a_1x^{n-1} + \cdots + a_n$

III. Penyelesaian Masalah

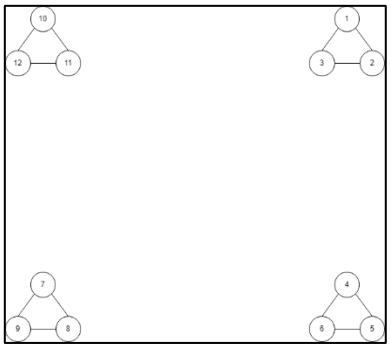
A. Deskripsi Masalah

Pertama-tama, penulis akan mencoba mengumpulkan informasi-informasi relevan untuk membuat deskripsi permasalahan yang lebih jelas. Informasi mengenai pengelompokkan mahasiswa pada tugas besar pertama dapat diakses pada tautan berikut : Pendataan Tubes 1 Algeo 24/25. Dari tautan tersebut, dapat dilihat bahwa terdapat 165 mahasiswa yang dibagai menjadi 55 Kelompok, dimana tiap kelompok berisi 3 orang. Dalam hal ini, kita dapat misalkan a_i, b_i, c_i adalah mahasiswa pada kelompok ke-i, terurut dari kelompok nomor pertama sampai dengan kelompok nomor terakhir. Lebih jelasnya, a_1 adalah Muhammad Luqman Hakim, b_1 adalah Zulfaqqar Nayaka Athadiansyah, c_1 adalah Muhammad Farell Athalla, a_2 adalah Lutfi Hakim Yusra, dan seterusnya. Dapat dilihat bahwa pemetaan tersebut adalah korespondensi satu-satu. Lebih lanjut, pemetaan tersebut mempermudah kita dalam menentukan apakah 2 mahasiswa A dan B dapat berada di kelompok yang sama untuk Tugas Besar 2. a_1 tidak bisa berkelompok dengan b_1 maupun c_1, a_2 tidak bisa berkelompok dengan b_2 maupun c_2, dan seterusnya.

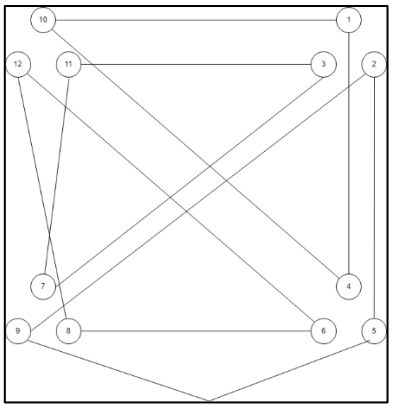
B. Representasi Permasalahan Dengan Graf

Permasalahan yang dihadapi dapat dideskripsikan dengan pemodelan graf.

Contoh pemodelan graf yang akan disajikan adalah kasus apabila jika hanya terdapat 4 kelompok, dimana anggota tiap kelompok adalah 3 orang.



Gambar 3.1. Pengelompokan pada Tugas Besar 1



Gambar 3.2. Pengelompokan pada Tugas Besar 2

Dapat dilihat bahwa dalam pemodelan graf, persoalan pengelompokan tugas besar dapat dimodelkan sebagai suatu permasalahan penambahan edge, dengan syarat bahwa edges tidak boleh ditambahkan pada 2 vertex yang sudah ada dalam graf pertama.

C. Solusi I - Rekursif

Secara umum, misalkan terdapat N kelompok, dengan anggota kelompok ke-i adalah $\{a_i, b_i, c_i\}$. Misalkan belum terdapat kelompok yang terbentuk untuk tugas besar 2. Untuk membentuk kelompok pertama, kita dapat memilih 3 bilangan berbeda, misalkan x, y, z, Dimana $x \neq y \neq z$. Anggota kelompok baru tersebut adalah 1 anggota pada kelompok ke-x, 1 anggota pada kelompok ke-y, serta 1 anggota pada kelompok ke-z. Pemilihan tersebut memastikan bahwa anggota kelompok baru tersebut tidak mungkin sekelompok pada tugas besar sebelumnya. Kita dapat melakukan iterasi ini terus menerus sampai semua mahasiswa mendapatkan kelompoknya. Perhatikan bahwa setelah kita melakukan suatu iterasi, anggota dari kelompok x, kelompok y, serta kelompok z akan berkurang sebanyak 1. Iterasi pembentukan kelompok tersebut akan diulangi sebanyak N kali, sehingga terbentuk N kelompok.

Lebih formilnya, proses pembentukan suatu pengelompokan yang valid dapat dilakukan sebagai berikut.

```
(for i = 1; i <= N; i++) {
  choose x, y, z, where x /= y and y /= z and x /= z
  and count(x) > 0 and count(y) > 0 and count(z) > 0

  a = one player from group x
  b = one player from group y
  c = one player from group z

  form group {a, b, c}
  count(x) = count(x) - 1
  count(y) = count(y) - 1
  count(z) = count(z) - 1
}
```

Gambar 3.3. Proses pembentukan kelompok yang valid

Perhatikan bahwa dalam setiap iterasi, permasalahan yang dihadapi adalah hal yang sama. Setiap iterasi hanya mengubah 1 hal, yaitu jumlah anggota pada suatu grup. Sehingga kita dapat memformulasikan suatu fungsi rekursi untuk menghitung total kemungkinan pada suatu konfigurasi count yang mungkin.

$$f(count) = \sum_{x,y,z \in valid} P \times f(count')$$

Dengan kasus basis dari rekursi tersebut adalah

- $f([0, 0, \dots, 0]) = 1$
- $f(count_{invalid}) = 0$
  $count_{invalid}$ adalah konfigurasi $count$ jika tidak ada kemungkinan untuk membentuk group yang memenuhi syarat. Atau dapat dikatakan, dalam konfigurasi tersebut, kita tidak bisa memilih (x, y, z) yang memenuhi syarat

  Beberapa contoh $count_{invalid}$ adalah $[0, 0, 1, 2]$, $[0, 3, 0, 0]$, $[0, 1, 2, 1]$.

Dimana :

- $count$ adalah array atau himpunan (dalam makalah ini akan dipanggil dengan array) yang menyatakan jumlah anggota tersisa pada masing-masing dari N kelompok awal
- (x, y, z) adalah triplet indeks kelompok berbeda yang dipilih untuk membentuk kelompok baru,
- '$valid$' berarti triplet (x, y, z) masih memiliki setidaknya 1 anggota tersisa di tiap kelompok tersebut (sehingga memungkinkan terbentuknya satu kelompok baru), Aturan tambahan adalah pemilihan x, y, z harus memenuhi aturan $x < y < z$, agar tidak terjadi *double counting.*
- $count'$ adalah konfigurasi (keadaan) baru, setelah masing-masing dari kelompok (x, y, z) diambil satu anggotanya (sehingga berkurang 1 di masing-masing kelompok tersebut).
- P adalah faktor kombinatorial ketika kita memilih satu orang pada group x, y, z. Lebih jelasknya, Banyaknya cara memilih satu anggota dari kelompok x, ditambah satu dari kelompok y, ditambah satu dari kelompok z adalah $count(x) \times count(y) \times count(z)$

Pada proses berikutnya, kita kemudian memanggil f($count'$) karena masalah "sisa pemilihan" yang kita hadapi tetap sama secara struktur: kita kembali perlu mengambil 3 kelompok berbeda, masing-masing memberi satu anggota, untuk membentuk kelompok baru berikutnya. Prinsip ini berlangsung rekursif sampai semua anggota pada setiap kelompok habis dipilih. Jawaban dari permasalahan pengelompokan Tugas Besar 2 adalah

$$f(Init)$$
$$Init = [3, 3, 3, \dots, 3]$$

Nilai dari array $Init$ adalah kondisi anggota pada kelompok pada awal pengelompokan, Dimana tiap kelompok beranggotakan 3 orang.

Berikut adalah sekilas simulasi bagaimana penghitungan apabila terdapat 4 kelompok :

$$f([3,3,3,3] = 27f([2,2,2,3]) + 27f([2,2,3,2]) + 27f([2,3,2,2]) + 27f([3,2,2,2])$$
$$f(2,2,2,3) = 8f([1,1,1,3]) + 12f([1,1,2,2]) + 12f([1,2,1,2]) + 12f([2,1,1,2])$$
$$f(1,1,1,3) = f([0,0,0,3]) + 3f([0,0,1,2]) + 3f([0,1,0,2]) + 3f([1,0,0,2])$$
$$f([0,0,0,3]) = 0$$
$$f([0,0,1,2]) = 0$$
$$f([0,1,0,2]) = 0$$
$$f([1,0,0,2]) = 0$$
$$\dots$$

Setelah semua perhitungan dilakukan, akan didapat bahwa

$$f[3,3,3,3] = (27 \times 12) + (27 \times 12) + (27 \times 12) + (27 \times 12) = 1296$$

Artinya, terdapat 1296 kemungkinan pengelompokan tugas besar 2, jika terdapat 4 kelompok

D. Solusi II – Optimisasi Dari Solusi I

Kendati jauh lebih cepat dibandingkan dengan teknik naïve *brute-force*, Solusi I masih dapat dioptimisasi lebih lanjut dengan observasi lebih lanjut. Perhatikan bahwa permasalahan ini simetris terhadap identitas kelompok. Artinya, tidak ada perbedaan antara "kelompok A" dan "kelompok B" jika mereka sama-sama memiliki sisa 2 orang, misalnya. Perbedaan label/grup mana pun dapat kita tukar, dan itu tidak mengubah jumlah cara keseluruhan untuk membentuk kelompok Tugas Besar 2. Dengak kata lain, untuk suatu konfigurasi $count_1$ dan $count_2$ dengan banyaknya kelompok dengan 0 anggota sama, banyaknya kelompok dengan 1 anggota sama, banyaknya kelompok dengan 2 anggota sama, serta banyaknya kelompok dengan 3 anggota sama, maka nilai dari $f(count_1) = f(count_2)$. Lebih formilnya adalah sebagai berikut :

- Misalkan $count_1$ dan $count_2$ adalah suatu konfigurasi sisa anggota pada tiap-tiap kelompok
- Misalkan $freq_0(count_1)$ adalah jumlah kelompok dengan 0 anggota pada konfigurasi $count_1$, begitu juga dengan $freq_1(count_1)$, $freq_2(count_1)$, $freq_3(count_1)$, $freq_0(count_2)$, $freq_1(count_2)$, $freq_2(count_2)$, $freq_3(count_2)$.
- Jika
$$freq_0(count_1) = freq_0(count_2)$$
$$freq_1(count_1) = freq_1(count_2)$$
$$freq_2(count_1) = freq_2(count_2)$$
$$freq_3(count_1) = freq_3(count_2)$$
maka berlaku
$$f(count_1) = f(count_2)$$

Hal tersebut dapat dilihat pada perhitungan sebelumnya, Dimana nilai dari $f([2,2,2,3]) = f([2,2,3,2]) = f([2,3,2,2]) = f([3,2,2,2]) = 12$.

Dengan adanya sifat tersebut, kita dapat memanfaatkan sifat tersebut untuk mengoptimasi Solusi I lebih lanjut, yaitu hanya dengan memanfaatkan $freq_0, freq_1, freq_2, freq_3$ dari suatu $count$. Lebih lanjut lagi, kita bisa mengabaikan $freq_0$, karena $freq_0$ merupakan banyaknya kelompok yang anggotanya sudah habis pada suatu konfigurasi. Sekarang, kita dapat menformulasikan suatu fungsi rekursi untuk menghitung banyak kemungkinan pengelompokkan, hanya dengan berdasarkan $freq_1, freq_2, freq_3$, yaitu :

$$g(freq_1, freq_2, freq_3) = \sum_{\substack{semua \\ pola \\ valid}} P \times g(freq_1', freq_2', freq_3')$$

Dimana :

- $freq_1'$ adalah banyaknya kelompok dengan banyak anggota 1 setelah dilakukan pengambilan
- $freq_2'$ adalah banyaknya kelompok dengan banyak anggota 2 setelah dilakukan pengambilan
- $freq_3'$ adalah banyaknya kelompok dengan banyak anggota 3 setelah dilakukan pengambilan
- "Pola Valid" adalah (i, j, k), dimana $0 \le i \le j \le k \le 3$, Artinya, suatu kelompok akan dibentuk dengan mengelompokkan 1 mahasiswa dari kelompok dengan banyak anggota i, 1 mahasiswa dari kelompok dengan banyak anggota j, dan 1 mahasiswa dari kelompok dengan banyak anggota k.
- P adalah factor kombinatorika ketika memilih suatu "Pola Valid". Sederhananya, P adalah banyak cara memilih mahasiswa dalam suatu kelompok yang memenuhi "Pola Valid".

Dalam penghitungan ini, nilai P terbilang cukup sulit untuk dihitung. Untuk menunjukkan bagaimana menghitung P, perhatikan kasus berikut. Misal terdapat suatu konfigurasi dengan

- $freq_1 = 4$, artinya terdapat 4 kelompok dengan tepat 1 anggota
- $freq_2 = 1$, artinya terdapat 1 kelompok dengan tepat 2 anggota
- $freq_3 = 6$, artinya terdapat 6 kelompok dengan tepat 3 anggota

$freq$ di atas merepresentasikan banyak $count$, salah satunya adalah $[1,1,2,2,2,3,3,3,3,3,3]$

Salah satu pola pengambilan yang valid adalah $(1,2,3)$.

- Banyak mahasiswa yang berada di dalam kelompok dengan 1 anggota = $4 \times 1 = 4$
- Banyak mahasiswa yang berada di dalam kelompok dengan 2 anggota = $1 \times 2 = 2$
- Banyak mahasiswa yang berada di dalam kelompok dengan 3 anggota = $6 \times 3 = 18$
- $P =$ banyak cara memilih mahasiswa dalam suatu kelompok yang memenuhi "Pola Valid", yakni $4 \times 2 \times 18 = 144$

Contoh pola pengambilan yang lain adalah $(2,3,3)$.

- Banyak mahasiswa yang berada di dalam kelompok dengan 2 anggota = $1 \times 2 = 2$
- Banyak mahasiswa yang berada di dalam kelompok dengan 3 anggota = $6 \times 3 = 18$
- Banyak mahasiswa yang berada di dalam kelompok dengan 3 anggota = $5 \times 3 = 15$. Perhatikan bahwa hanya terdapat 5 kelompok, karena mahasiswa tidak boleh berasal dari kelompok yang sama.
- $P =$ banyak cara memilih mahasiswa dalam suatu kelompok yang memenuhi "Pola Valid", yakni $\frac{2 \times 18 \times 15}{2} = 270$. Perhatikan bahwa perlu dibagi 2, karena urutan pengambilan mahasiswa tidak diperhatikan.

Jawaban dari permasalahan pengelompokan Tugas Besar 2 adalah $g(0,0,N)$, karena kondisi awal pengelompokan adalah $Init = [3,3,3,\ldots,3]$, yaitu terdapat N kelompok dengan anggota 3 orang.

IV. IMPLEMENTASI

A. Implementasi Solusi I - Rekursif

Implementasi dilakukan dengan Bahasa Python. Alasan digunakannya Bahasa pemrograman python tersebut adalah karena penghitungan yang bisa sangat besar, sehingga diperlukannya suatu Bahasa yang dapat melakukan kalkulasi terhadap bilangan besar. Python adalah salah satu Bahasa pemrograman yang dapat melakukan kalkulasi tersebut.

```python
# Memoization dictionary for storing the results of states
memo = {}

# Recursive function to calculate the number of ways to form groups
def solve_config(count_group):
    # Check if all groups are empty
    if all(x == 0 for x in count_group):
        return 1

    # Check memo
    tuple_state = tuple(count_group)
    if tuple_state in memo:
        return memo[tuple_state]

    ways = 0
    N = len(count_group)

    # Try selecting 3 different groups (x < y < z) that still have members
    for x in range(N):
        if count_group[x] == 0:
            continue
        for y in range(x + 1, N):
            if count_group[y] == 0:
                continue
            for z in range(y + 1, N):
                if count_group[z] == 0:
                    continue

                # Number of ways to pick 1 member from groups x, y, z
                pick_count = count_group[x] * count_group[y] * count_group[z]

                # Reduce 1 member from each group and proceed recursively
                count_group[x] -= 1
                count_group[y] -= 1
                count_group[z] -= 1

                ways += pick_count * solve_config(count_group)

                # Backtrack to restore the state
                count_group[x] += 1
                count_group[y] += 1
                count_group[z] += 1

    # Store the result in memo and return
    memo[tuple_state] = ways
    return ways
```

Gambar 4.1. Implementasi Solusi dalam python

```
Enter the number of groups: 5
Possibilities:  132192
Time taken: 1.03 ms
```

Gambar 4.2. Kode Ketika dijalankan

B. Implementasi Solusi II – Optimisasi Dari Solusi I

Implementasi dilakukan dengan Bahasa Python. Alasan digunakannya Bahasa pemrograman python tersebut adalah karena penghitungan yang bisa sangat besar, sehingga diperlukannya suatu Bahasa yang dapat melakukan kalkulasi terhadap bilangan besar. Python adalah salah satu Bahasa pemrograman yang dapat melakukan kalkulasi tersebut

```python
# Precomputed factorials
factorials = [1] * 100
for i in range(1, 100):
    factorials[i] = factorials[i - 1] * i

# Predefined use array
use = [
    [0, 0, 0],
    [1, 1, 1],
    [2, 2, 2],
    [0, 0, 1],
    [0, 0, 2],
    [0, 1, 1],
    [1, 1, 2],
    [0, 2, 2],
    [1, 2, 2],
    [0, 1, 2],
]
```

Gambar 4.3. Tahap Inisiasi

Potongan kode tersebut menunjukkan komputasi awal untuk factorial, serta deklarasi array “use”. Array tersebut merupakan semua pola valid yang dapat digunakan untuk penghitungan selanjutnya.

```python
def solve(a, b, c):
    if a == 0 and b == 0 and c == 0:
        return 1

    ret = 0
    for l in range(10):
        arr = [a, b, c]
        valid = True

        # Update arr based on use[l]
        for m in range(3):
            arr[use[l][m]] -= 1

        # Check validity
        for x in arr:
            if x < 0:
                valid = False

        # Undo part of the decrement for repeated indices
        for m in range(3):
            if use[l][m] > 0:
                arr[use[l][m] - 1] += 1

        if valid:
            mul = 1
            div = 1
            if use[l][0] == use[l][1] == use[l][2]:
                div = factorials[3]
            elif use[l][0] == use[l][1] or use[l][1] == use[l][2]:
                div = factorials[2]

            num = [a, 2 * b, 3 * c]
            for m in range(3):
                mul *= num[use[l][m]]
                num[use[l][m]] -= (use[l][m] + 1)

            ret += (mul * solve(arr[0], arr[1], arr[2])) // div

    return ret
```

Gambar 4.4. Implementasi Solusi dalam python

```
Enter the number of groups: 55
Possibilities:  893860391271184476173227619234527221374383067262809272988438495842956981
827462356407646820075699122915065902090211493262972721701299303431222432252587681343128617045389702416302080
Time taken: 140.55 ms
```

Gambar 4.5. Kode Ketika dijalankan

C. Komparasi Runtime dan Output Solusi

Pada bagian ini, akan dilakukan komparasi dengan 3 solusi, yakni

- Solusi *Naïve Bruteforce*, selanjutnya akan disebut Solusi NB
- Solusi Rekursif, selanjutnya akan disebut Solusi R
- Solusi R yang dioptimasi, selanjutnya akan disebut Solusi Opt R

Hasil pengujian output dapat dilihat pada table berikut :

No.	Input (N)	Output Solusi NB	Output Solusi R	Output Solusi Opt R
1	4	1296	1296	1296
2	5	132192	132192	132192

3	6	19258560	19258560	19258560
4	7	Unavailable*	3829057920	3829057920
5	8	Unavailable*	1001695548672	1001695548672
6	9	Unavailable*	3339731150 62272	3339731150 62272
8	10	Unavailable*	1383486452 13579264	1383486452 13579264
9	100	Unavailable*	Unavailable *	Sekitar 6.6×10^{377}

Tabel 4.1. Hasil Pengujian Output
Unavailable* = Waktu yang diperlukan program untuk komputasi terlalu besar

Hasil pengujian *runtime* dapat dilihat pada tabel berikut :

No.	Input (N)	Runtime Solusi NB	Runtime Solusi R	Runtime Solusi Opt R
1	4	9.69 ms	1.00 ms	0.00 ms
2	5	840.8 ms	1.01ms	0.00ms
3	6	118377 ms 118 seconds ~2 minutes	9.02 ms	0.00 ms
4	7	Unavailable*	73.87 ms	0.00 ms
5	8	Unavailable*	496.29 ms	0.99 ms
6	9	Unavailable*	3197.69 ms	1.00 ms
7	10	Unavailable*	19291.67 ms	1.01 ms
8	100	Unavailable*	Unavailable*	901.46 ms

Tabel 4.2. Hasil Pengujian *Runtime*
Unavailable* = Waktu yang diperlukan program untuk komputasi terlalu besar

D. Analisis Kompleksitas Waktu

Pada Solusi R, program tersebut menyelesaikan masalah pengelompokan dengan pendekatan rekursi dengan *memoisasi*, dimana setiap state adalah sautu array $count$, yang merepresentasikan jumlah anggota dari tiap tiap kelompok.

Untuk N kelompok :

- Kemungkinan element pertama $count$ adalah $\{0, 1, 2, 3\}$, terdapat 4 kemungkinan
- Kemungkinan element kedua $count$ adalah $\{0, 1, 2, 3\}$, terdapat 4 kemungkinan
- ...
- Kemungkinan element ke$-N$ $count$ adalah $\{0, 1, 2, 3\}$, terdapat 4 kemungkinan

Sehingga, banyak kemungkinan *state* untuk array $count$ adalah 4^N.

Disamping itu, untuk setiap $state$, terdapat *looping untuk* $x \in [1 \dots N], y \in [x+1 \dots N], z \in [y+1 \dots N]$. Sehingga, kurang lebih akan terdapat $\frac{(N)(N-1)(N-2)}{3!}$ operasi untuk setiap *state.* Sehingga, total operasi yang dilakukan, kurang lebihnya adalah

$$T(N) = 4^n \times \frac{(N)(N-1)(N-2)}{3!}$$

Dengan demikian, big-O untuk program tersebut adalah $O(4^n \times N^3)$

Pada Solusi R Opt, Program tersebut menyelesaikan masalah pengelompokan menggunakan pendekatan rekursif dengan *memoisasi*, di mana setiap state direpresentasikan oleh tiga parameter a, b, dan c. Parameter tersebut menyatakan jumlah kelompok dengan 1, 2, dan 3 anggota tersisa. Kompleksitas waktu program ini dapat dianalisis dengan menguraikan ruang pencarian, jumlah operasi per state, dan pengaruh memoization terhadap kinerja keseluruhan.

Fungsi $solve(a, b, c)$ mengeksplorasi semua kemungkinan kombinasi state yang dapat terjadi untuk jumlah total mahasiswa $3N$ yang dikelompokkan ke dalam kelompok dengan 1, 2, atau 3 anggota. Di samping itu, a menyatakan banyak kelompok dengan 1 anggota, b menyatakan banyak kelompok dengan 2 anggota, serta c menyatakan banyak kelompok dengan 3 anggota. Oleh karena itu, $state$ yang akan ditelusuri oleh program hanyalah *state* yang memenuhi syarat

$$a + 2b + 3c \leq 3N$$

Atau, bisa diaproksimasi lebih lanjut

$$a + b + c \leq N$$

Banyaknya (a, b, c) yang memenuhi syarat tersebut dapat dihitung dengan *stars and bars theorem:*

- $a + b + c = 0$
  Banyak tuple yang memenuhi = $\binom{2}{2}$
- $a + b + c = 2$
  Banyak tuple yang memenuhi = $\binom{3}{2}$
- $a + b + c = 3$
  Banyak tuple yang memenuhi = $\binom{4}{2}$
- ...
- $a + b + c = N$
  Banyak tuple yang memenuhi = $\binom{N+2}{2}$

Maka total state yang perlu dikunjungi adalah jumlah semua kasus diatas, yakni $\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{N}{2}$.

Berdasarkan *Hockey-Stick Identity*,

$$\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$$

Maka, banyaknya state yang perlu dikunjungi adalah

$$\binom{2}{2} + \binom{3}{2} + \binom{4}{2} + \cdots + \binom{N}{2} = \sum_{i=2}^{N} \binom{i}{2} = \binom{N+1}{3}$$

Kemudian, untuk setiap state yang dikunjungi, terdapat *looping* untuk $l \in [0..9]$ dan $m \in [0 \dots 2]$, sehingga kurang lebih terdapat 10×3 operasi untuk setiap state. Sehingga, total operasi yang dilakukan, kurang lebihnya adalah

$$T(N) = 30 \times \binom{N+1}{3}$$

Dengan demikian, big-O untuk program tersebut adalah $O(N^3)$

V. KESIMPULAN

Permasalahan menghitung total Kemungkinan Pengelompokan pada Tugas Besar 2 Aljabar Linear dan Geometri merupakan permasalahan kombinatorik yang memerlukan eksplorasi menyeluruh terhadap kemungkinan pembentukan kelompok. Terdapat 3 pendekatan yang telah

dibahas, yakni pendekatan naïve *bruteforce*, rekursi, serta rekursi yang dioptimasi. Akan tetapi, Solusi *naïve bruteforce* dan rekursi masih memiliki kompleksitas waktu eksponensial, menyebabkan runtime yang sangat Panjang, Dimana untuk runtime untuk input $N = 6$, Solusi *naïve bruteforce* telah menyentuh angka 2 menit, sementara untuk Solusi rekursi sendiri, untuk runtime dari input $N = 10$, Solusi tersebut telah menyetuh angka ~ 19 detik. Di sisi lain, Solusi rekursi yang telah dioptimasi berhasil menyelesaikan permasalahan dengan kompleksitas waktu non-exponential, yaitu $O(N^3)$. Jawaban akhir dari permasalahan ini, yakni total kemungkinan pengelompokan pada Tugas Besar 2 Aljabar Linear dan Geometri, adalah
893860391271184476173227619234527221374383067262809272988438495842956981827462356407646820075699122915065902090211493262972721701299303431222432252587681343128617045389702416302 08, atau sekitar

$$8.938 \times 10^{178}$$

VI. Referensi

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf diakses pada 8 Januari 2024.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian2-2024.pdf diakses pada 8 Januari 2024.

[3] Chuan-Chong, C., Khee-Meng, K., "Principles and Techniques in Combinatorics", Singapore: World Scientific, 1992, ch. 1-4.

[4] CH Jones (1996) Generalized Hockey Stick Identities and N-Dimensional Block Walking. Fibonacci Quarterly 34(3), 280-288. B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[5] Merris, Russell (2003). Combinatorics (2nd ed.). Hoboken, N.J.: Wiley-Interscience. p. 45. ISBN 0-471-45849-X. OCLC 53121765.

VII. Lampiran

Program yang digunakan pada makalah ini disimpan pada tautan github : https://github.com/AkuJanjiTidakAkanRasisLagi/MakalahMatdis

VIII. Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 8 Januari 2025

Mochammad Fariz Rifqi Rizqulloh, 13523069